На этом занятии мы с вами рассмотрим пример использования статических переменных и методов класса для реализации известного паттерна Singleton (синглтон).

Смысл этого паттерна прост. **Объявляется класс, так, чтобы от него мог существовать ровно один объект в один момент времени**. Сделаем это следующим образом. Объявим класс, который пусть так и называется – Singleton. И в этом классе разместим статическое поле – указатель на возможный объект этого класса, а также некие целочисленные данные:

```language-cpp
class Singleton {
    int data {0};
    static Singleton* instance_ptr;
};
```

Изначально значение статического поля (переменной) instance_ptr установим в значение nullptr:

```language-cpp
Singleton* Singleton::instance_ptr = nullptr;
```

Это будет означать, что класс не имеет ни одного объекта.

Я, думаю, вы понимаете, почему мы указатель instance_ptr на объект определили статическим? Да, **он не должен относиться к самим объектам класса (лишь ссылаться на один, при его наличии) и быть определенным в области видимости класса, так как непосредственно связан с логикой паттерна Singleton**. Кроме того, статическое поле задается сразу в момент выполнения программы (до формирования каких-либо объектов класса) и в нашем случае принимает начальное значение nullptr.

Далее, мы запретим создание объектов класса напрямую, поэтому разместим конструктор по умолчанию в приватной секции. Также запретим создание копий объектов, поэтому удалим встроенный конструктор копирования. Наконец, объявим свой деструктор класса, в котором статическое поле instance_ptr снова установим в значение nullptr для возможности создания нового объекта в случае удаления прежнего. В итоге, класс Singleton примет вид:

```language-cpp
class Singleton {
    int data {0};
    static Singleton* instance_ptr;
```

```
    Singleton()
        { }
public:
    Singleton(const Singleton& ) = delete;
    ~Singleton() { instance_ptr = nullptr; }
};
```

Хорошо, но спрашивается, как мы будем создавать объект этого класса, если единственный доступный конструктор находится в секции private? Очень просто! Мы объявим в классе Singleton специальный статический метод для создания единственного объекта. Назовем этот метод get_instance и определим его следующим образом:

```language-cpp
    static Singleton* get_instance()
    {
        if (instance_ptr == nullptr) {
            instance_ptr = new Singleton();
        }

        return instance_ptr;
    }
```

Работа его вполне очевидна. Если статическое поле instance_ptr равно nullptr, значит, ни одного объекта класса не существует, и он создается. В конце возвращается указатель instance_ptr.

Почему этот метод мы сделали статическим? Очевидно, **чтобы можно было его вызывать напрямую из класса, не имея ни одного объекта, следующим образом**:

```language-cpp
int main()
{
    Singleton* ptr = Singleton::get_instance();

    std::cout << ptr << std::endl;

    delete ptr;

    return 0;
}
```

В теле метода get_instance осуществляется обращение к приватному статическому полю instance_ptr и при необходимости создается ровно один объект класса.

Если мы попробуем создать еще один объект:

```language-cpp
Singleton* ptr2 = Singleton::get_instance();
```

то указатель ptr2 будет ссылаться на тот же самый объект, второго создано не будет.

Ну и в конце описания класса объявим в нем еще два метода для работы с приватной переменной data:

```language-cpp
    void set_data(int d) {data = d;}
    int get_data() { return data; }
```

В итоге у нас с вами получился следующий класс, реализующий паттерн синглтон:

```language-cpp
class Singleton {
    int data {0};
    static Singleton* instance_ptr;

    Singleton()
        { }
public:
    Singleton(const Singleton& ) = delete;
    ~Singleton() { instance_ptr = nullptr; }

    static Singleton* get_instance()
    {
        if (instance_ptr == nullptr) {
            instance_ptr = new Singleton();
        }

        return instance_ptr;
    }

    void set_data(int d) {data = d;}
    int get_data() { return data; }
};

Singleton* Singleton::instance_ptr = nullptr;
```

А использовать его можно, например, так:

```language-cpp
int main()
{
```

```
    Singleton* ptr = Singleton::get_instance();
    ptr->set_data(1);
    Singleton* ptr2 = Singleton::get_instance();

    std::cout << ptr << " " << ptr2 << std::endl;
    std::cout << ptr2->get_data() << std::endl;

    delete ptr;
    return 0;
}
```

Видим, что оба указателя содержат одинаковый адрес, и метод get_data возвращает ранее записанное значение 1.

Вот так, достаточно просто, можно на C++ реализовать паттерн Singleton, используя статические поля и методы, а также правильно настраивая все его конструкторы.